EasyChair Preprint

№ 15499

# LSM-Tree Based Hardware System Design for Compaction Acceleration

Sirui Peng, Lin Yang, Zhikang Ding, Xingyu Chen, Xu Kong, Haidong Tian, Xiankui Xiong, Xiaoyong Xue and Xiaoyang Zeng



November 29, 2024

# LSM-tree Based Hardware System Design for Compaction Acceleration

Sirui Peng[1], Lin Yang[1], Zhikang Ding[1], Xingyu Chen[1], Xu Kong[1], Haidong Tian[2], Xiankui Xiong[2], Xiaoyong Xue[1]*, Xiaoyang Zeng[1]
[1]School of Microelectronics, State Key Laboratory of Integrated Chips and Systems, Fudan University, Shanghai, China
[2]State Key Laboratory of Mobile Network and Mobile Communication Multimedia Technology, ZTE, Shenzhen, China
*Email of corresponding author: xuexiaoyong@fudan.edu.cn

***Abstract*—With the rapid development of technology and the advent of the era of big data, Log-structured merge tree (LSM-tree) based key-value storage systems are widely used in data centers that persistently store ultra-large-scale data. This paper proposes offloading the compaction operation to FPGA for acceleration, which can free up CPU computational resources and I/O bandwidth, thus improving the quality of service of the overall system. To evaluate the acceleration performance of the compaction module on hardware, this paper also completes the design of the NAND Flash controller and the corresponding storage card. The experimental results show that the compaction module implemented in this work can achieve up to 11.07x acceleration performance compared to LevelDB.**

***Keywords—LSM-tree, Compaction, NAND Flash, FPGA***

## I. INTRODUCTION

With the rapid development of machine learning, intelligent computing, and cloud databases, storing massive data is becoming increasingly important[1]. Traditional relational databases are no longer suitable for highly concurrent and low-latency access scenarios to unstructured data[2]. Therefore, Key Value (KV) storage systems have emerged and become an important part of data storage facilities for data-intensive applications due to their simplicity, high throughput and scalability [3-5]. The LSM-tree-based KV storage system has a very significant module, compaction, which is designed to merge multiple sorted string tables (SSTables). The compaction operation is a very host resource-consuming operation [6], which reads a large number of relevant SSTables from the SSD into the host memory first, and writes them back to the SSD after processing. This process takes up a large amount of CPU computational resources and I/O bandwidth, and results in severe read/write amplification, which leads to the user's Quality of Service (QoS) degradation. Especially for large granularity KV data, the large amount of data handling and processing makes this effect more severe [7].

To reduce the consumption of host resources and I/O bandwidth, using heterogeneous hardware acceleration is an effective optimization method. A common practice is to use the embedded CPU in FPGA to accelerate operations such as compaction or indexing in LSM-tree. Sun X [8], Zhang T [9] and Lim M [10] et al. offloaded the compaction operation in LSM-tree to FPGA to accelerate the operation execution, which makes the CPU free from the heavy task of compaction calculation. In addition, the compaction operation does not need to read the SSTable data into memory and write it back after merging, which can greatly reduce the I/O bandwidth consumption. In particular, separating the compaction module from the host and designing a test platform for evaluation are crucial points to realize and verify the acceleration performance.

To reduce the resource and computational burden of the host and improve the QoS of the overall system, this paper proposes offloading the compaction operation into the FPGA and using hardware resources to realize it. In order to evaluate the effectiveness of this work in the real world, we also built a complete evaluation process to verify the performance improvement of this design. This paper makes the following contributions:

- The compaction acceleration module is implemented by using hardware circuits, which reduces the consumption of host resources and read/write amplification.
- A NAND flash controller is implemented to interact upper-level commands with the NAND flash chip, which is essential for the test platform.
- A hardware and software co-processing test platform is built for our designed structure, which verified the superiority of the work.

The rest of the paper is organized as follows. We discuss the hardware design of the compaction module in Section II. In Section III, we discuss the simulation platform. Section IV presents the results of the compaction module performance. Finally, we conclude the paper by summarizing the key points in Section V.

## II. COMPACTION MODULE DESIGN

### A. Overall Structure of Compaction

The overall structure of the compaction acceleration module is shown in Figure 1. The compaction module is functionally divided into three parts: the decoder module, the comparer_merger module, and the encoder module.

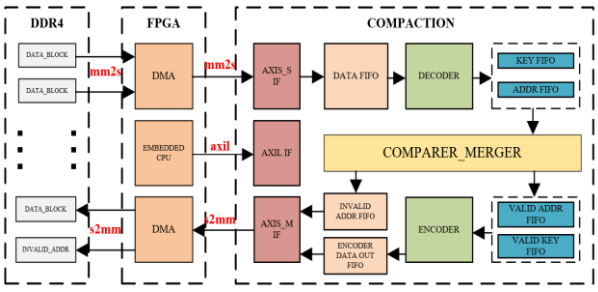


Fig. 1. The overview of the compaction module.

The decoder module decodes the input data_block, converts it into key data and address data, and stores it in the corresponding first-input-first-output (FIFO). The comparer _merger module takes the key data and the address data from the FIFO, processes the format of the key data, performs a dictionary order comparison and outputs the valid key data,

This work was supported in part by the National Key R&D Program under Grant 2023YFB4404700, in part by the National Natural Science Foundation of China under Grant 62274038, in part by the Science and Technology Commission of Shanghai Municipality under Grant 21TS1401200 and Grant 22ZR1407100, and in part by State Key Laboratory of Integrated Chips and Systems under Grant SKLICS-Z202315.

the address data corresponding to the valid key data, and the invalid address data for garbage collection. The encoder module encodes the valid key and address data output by the comparer_merger module and converts them to 32-bit data, which is finally written into the encoder_data_out_fifo module.

### B. Global Register Definitions

The compaction acceleration module contains a global register module that is connected to the embedded CPU via the AXI-Lite bus. There are six 32-bit registers defined in the global register module, all of which are of read-write type. The global registers are illustrated in the table I.

TABLE I. GLOBAL REGISTER USED IN COMPACTION MODULE

| offset | register | description |
|---|---|---|
| 0x0000_0000 | REG_ENCODER_DATA_OUT_CNT | Indicates the amount of data written to the encoder_data_out_fifo module after encoding |
| 0x0000_0004 | REG_INVALID_ADDR_OUT_CNT | Indicates the number of invalid addresses output by the comparer_merger module. |
| 0x0000_0008 | REG_KEY0_ENTRY_IN_CNT | Indicates the amount of data carried from DDR4 to key0 entry by AXI-DMA. |
| 0x0000_000C | REG_KEY1_ENTRY_IN_CNT | Indicates the amount of data carried from DDR4 to key1 entry by AXI-DMA. |
| 0x0000_0010 | REG_COUNT_CONTROL | Controls the start and end of the reg_throughput_counter_module |
| 0x0000_0014 | REG_THROUGHPUT_COUNTER | Embedded software code can perform throughput calculations by reading this register value. |

### C. Decoder module

The function of the decoder module is to decode the data of the data_block module moved from the DDR4 by the AXI-DMA, convert it into key data and address data, and write it to the corresponding FIFO. This work separates the key and value components by storing <key, address> and <address, value> separately, where the address represents the address of the value data. The <key, address> pairs are stored in the space of the LSM-tree, while the <value> is written to the SSD using append write. Therefore, we can utilize FPGA to accelerate the compaction operation only for the separated <key, address>. This design reduces the data volume for compaction operations and optimizes the read/write performance of the LSM-tree by reducing read/write amplification. The schematic diagram of <key, value> separation is shown in Figure 2.

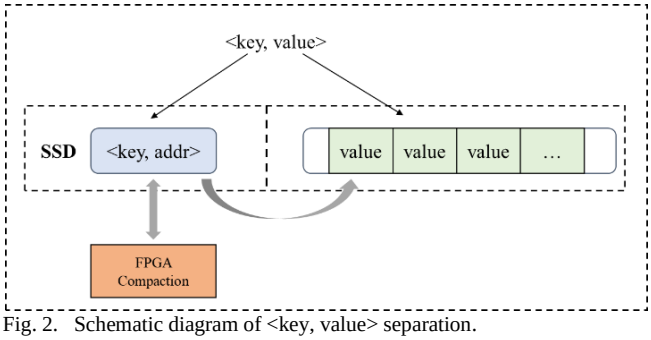


Fig. 2. Schematic diagram of <key, value> separation.

We illustrate the process of decoding in Figure 3. The key $entry_0$ is the start point key of the group, and the shared key length is 0, so the decoded key value is the value of the unshared key (i.e. "hellodb"). The shared key length of key $entry_1$ is 5, which means it shares 5 bytes of the previous key(i.e., "hello"), and combines it with its own unshared key, "world", to get the value of $key_1$ as "helloworld". The shared key length of key $entry_2$ is 8, (i.e., "hellowor"), which is combined with the unshared key of this entry to get the value of $key_2$ as "helloworld0". For a key entry with a vir_bit value of 1, it means that the entry is not output as a real key, but only as a shared key reference for the next key entry. Although a key entry with a vir_bit value of 1 is virtual, it is still treated as a complete key entry and goes through the normal decoding process. Note that in this example, key $entry_2$ and key $entry_3$ have the same prefix "helloworld" with a total of 10 bytes, but this design specifies that the maximum SHARED key length is 8, so the portion greater than 8 can only be reflected in the unshared key. The purpose of choosing prefix compression is to reduce the amplification of space, and continuing to increase the upper limit of shared key length will only increase the number of stages of the selector circuits in the decoder and encoder modules, which will not further optimize the space utilization.

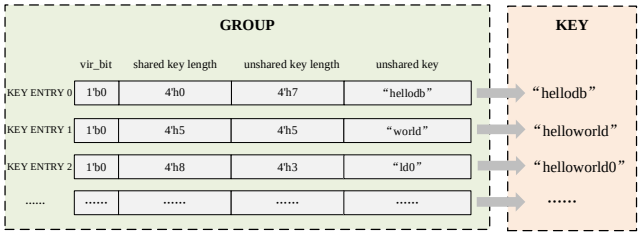


Fig. 3. The process of decoding.

### D. Comparer_Merger Module

The comparer_merger module is responsible for processing the format of the key data which is passed from the FIFO module and performs a dictionary order comparison. The comparer_merger module outputs the valid key data in dictionary order from smallest to largest and outputs the corresponding valid address. The comparer_merger module also writes the invalid address to the invalid_addr_fifo for later garbage collection.

We illustrate the process of comparing and merging in Figure 4. The $key_0$_fifo and $key_1$_fifo store the ordered key data and the overlapping key data in the two key_fifos are "3", "4", "5", and "6". The $addr_0$_fifo and $addr_1$_fifo store the key's corresponding address data. The comparer_merger module merges the identical key data and arranges the non-identical key data in the dictionary order from smallest to largest. After processing, the comparer_merger module writes them and their corresponding address into the valid_key_fifo and valid_addr_fifo respectively.

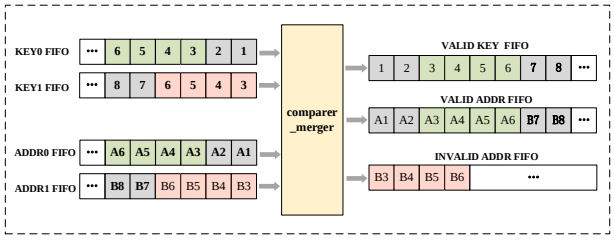


Fig. 4. The process of comparing and merging.

### E. *Encoder Module*

The function of the encoder module is to merge and encode the valid key data and valid address data by prefix compression output from the comparer_merger module and convert it to 32-bit wide data, which is finally written to the encoder_data_out_fifo. In order to be compatible with the decoder module, the encoder module needs to encode the input 133-bit wide key data into 64-bit data consisting of address, vir_bit, shared_key_len, unshared_key_len, and unshared_key, which will be converted into 32-bit data by using a data bit-width converter. Based on the characteristics of the key data to be encoded and the time sequence of the input, this design classifies the encoding types into three types: non-virtual key encoding; virtual key encoding; and group's last key encoding. There are differences in data processing between different encoding types, but the encoded 64-bit data conforms to the format shown in Figure 5.

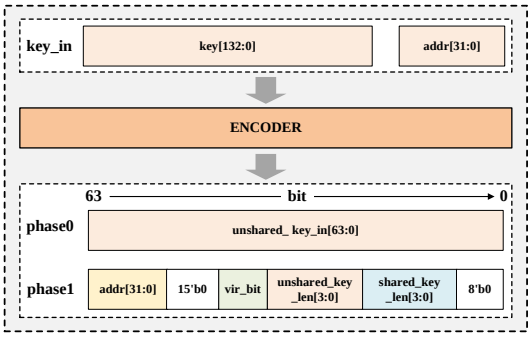


Fig. 5. The function of the encoder module

When the input key's unshared_key_len is less than or equal to 8 bytes, then the prefix is encoded in non-virtual key encoding mode. The encoding process of this mode is just the opposite of the decoding process in Figure 3. The purpose of the virtual key encoding mode is to solve the problem that 64-bit data cannot fully store an unshared key when the unshared_key_len of the key is greater than 8 bytes. A key with a shared_key_len greater than 8 is encoded multiple times. We treat the preceding entry as the shared portion of the following and set the vir_bit of the preceding entry to 1, which optimizes the speed of key queries. Note that during decoding, if vir_bit is 1, decoder_module does not process the key entry as a separate key entry, but rather considers it to be the prefix of the subsequent key entry. The process of virtual key encoding mode is shown in Figure 6. The group's last key encoding mode is used to handle the situation when the last key entry of a group is a virtual key entry. The encoder will mark the last key entry of the last group as an invalid entry and store the generated multiple key entries in the next group.

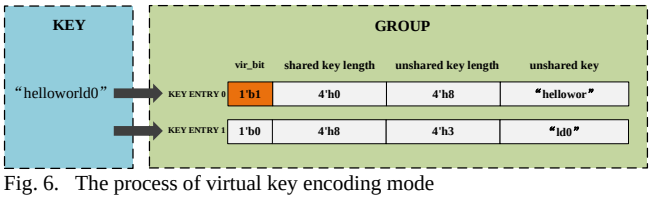


Fig. 6. The process of virtual key encoding mode

## III. SIMULATION PLATFORM

### A. *NAND Flash Controller*

In order to build a complete test platform, we must implement the NAND flash controller module, which is responsible for the interaction of upper-level commands with NAND Flash chip. The structure of NAND flash controller is shown in Figure 7. The NAND Flash interface can be divided into three layers: command classification layer, time sequence layer and physical interface layer. The command classification layer splits and combines the states of the seven commands that we have designed. The time sequence layer controls the timing of each signal line of the flash chip according to the timing requirements of each command. The physical interface layer implements functions such as cross-clocking domains, serial-to-parallel conversion, phase modulation and sampling. The NAND flash interface is externally connected to global registers and a DMA, which is responsible for command interaction with the embedded CPU and transferring data from the on-board DDR to the NAND flash respectively.

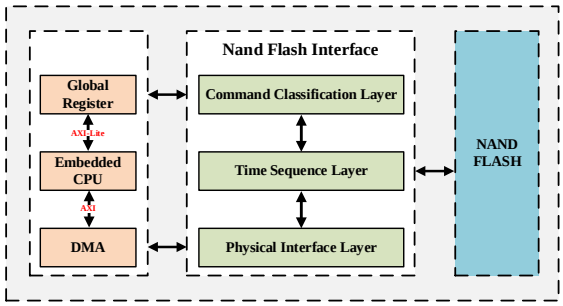


Fig. 7. The structure of NAND flash controller.

### B. *Overall Structure of Test Platform*

The test platform adopts hardware and software co-design, which is built on a Xilinx Kintex UltraScale FPGA KCU105. The structure of test platform is shown in Figure 8.

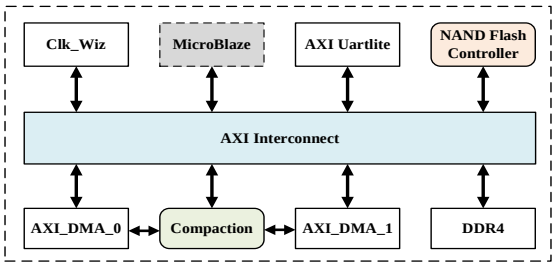


Fig. 8. The structure of the test platform

This design uses the MicroBlaze softcore as the CPU of the embedded system, which is responsible for running the embedded software code. The flow of the main function operation is shown in Figure 9.

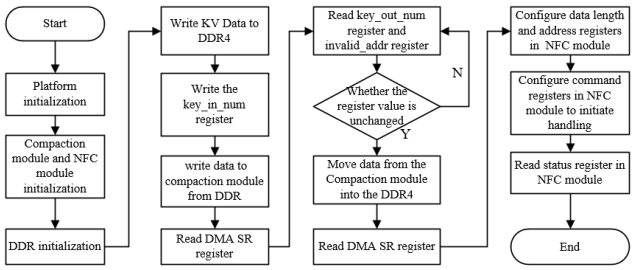


Fig. 9. The flow of embedded software main function code

AXI interconnect is responsible for data transfer between multiple IP cores. AXI_DMA is a high-performance DMA controller provided by Xilinx that enables programmable high-bandwidth data transfers, making data transfers between peripherals and memory more efficient. AXI Uartlite module

is used to print relevant information in the embedded software code for code debugging. The compaction module and NAND flash controller module are customized IPs that we have designed. The resource utilization of this test platform is shown in Table II.

TABLE II. RESOURCE UTILIZATION

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 44385 | 242400 | 18.31 |
| FF | 50980 | 484800 | 10.52 |
| BRAM | 433 | 600 | 72.17 |
| DSP | 3 | 1920 | 0.16 |
| IO | 136 | 520 | 26.15 |
| MMCM | 3 | 10 | 30 |
| PLL | 3 | 20 | 15.00 |

## IV. EVALUATION

We tested and analyzed the compaction performance of LevelDB [11], whose version is 1.23. The host is powered by a 6-core Intel(R) Core(TM) i5-8500 CPU @ 3.00GHZ processor and equipped with DDR4-2666 8GiB single-channel memory and Samsung 860 evo 500GB SSD. We calculate the throughput of the compaction module that we designed by reading the value of reg_throughput_counter through the embedded software. We tested 5 types of key entry quantities for evaluation and the results are shown in the table III.

TABLE III. PERFORMANCE TEST RESULTS OF COMPACTION MODULE

| Key number | Byte number | Throughput counter value | Processing efficiency (micros/entry) | Performance comparison with LevelDB [9] |
|---|---|---|---|---|
| 512 | 8192 | 9334 | 0.182305 | 6.54× |
| 1024 | 16384 | 14814 | 0.144668 | 8.24× |
| 2048 | 32768 | 26248 | 0.128164 | 9.30× |
| 4096 | 65536 | 45065 | 0.110022 | 10.84× |
| 8192 | 131072 | 88247 | 0.107723 | 11.07× |

As the number of key entries increases, the processing efficiency of the compaction module for acceleration increases. This is because when the number of input key entries is small, reading and writing registers on the software side take up most of the time due to the performance limitations of the MicroBlaze softcore. As the number of key entries increases, the real processing efficiency of the compaction module that we designed by hardware is better reflected. When the number of key entries is 8192, the processing efficiency of the compaction module is 11.07 times of LevelDB [11]. This fully demonstrates that utilizing hardware resources to implement the compaction module for acceleration has significant performance improvement.

## V. CONCLUSION

This paper proposes that offloading the compaction operation to FPGA and implementing it by using hardware resources can release the computational resources of CPU and I/O bandwidth, which will improve the QoS of the overall system. In order to evaluate the acceleration effect of the compaction module in hardware, this paper also accomplishes a complete test platform by hardware and software co-design. The experimental results show that the compaction module achieve up to 11.07x acceleration performance compared to LevelDB [11].

## REFERENCES


[1] Berisha, B., Mëziu, E. & Shabani, I. Big data analytics in Cloud computing: an overview. J Cloud Comp 11, 24 (2022). doi: 10.1186/s13677-022-00301-w.

[2] Nathan Marz and James Warren. 2015. Big Data: Principles and best practices of scalable realtime data systems (1st. ed.). Manning Publications Co., USA.

[3] Chen Luo and Michael J. Carey. 2019. LSM-based storage techniques: a survey. The VLDB Journal 29, 1 (Jan 2020), pp. 393–418.

[4] Y. Qiu *et al*., "FULL-KV: Flexible and Ultra-Low-Latency In-Memory Key-Value Store System Design on CPU-FPGA," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 8, pp. 1828-1444, 1 Aug. 2020, doi: 10.1109/TPDS.2020.2973965.

[5] T. Bisson, K. Chen, C. Choi, V. Balakrishnan and Y. -s. Kee, "Crail-KV: A High-Performance Distributed Key-Value Store Leveraging Native KV-SSDs over NVMe-oF," *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*, Orlando, FL, USA, 2018, pp. 1-8, doi: 10.1109/PCCC.2018.8710776.

[6] Hui Sun, Bendong Lou, Chao Zhao, Deyan Kong, Chaowei Zhang, Jianzhong Huang, Yinliang Yue, and Xiao Qin. 2023. An Asynchronous Compaction Acceleration Scheme for Near-Data Processing-enabled LSM-Tree-based KV Stores. ACM Trans. Embed. Comput. Syst. Just Accepted (September 2023). Doi: 10.1145/3626097.

[7] Lu L, Pillai T S, Gopalakrishnan H, et al. WiscKey: Separating Keys from Values in SSD-Conscious Storage[J]. ACM Transactions on Storage, 2017, 13(1):1-28. doi: 10.1145/3033273

[8] X. Sun, J. Yu, Z. Zhou and C. J. Xue, "FPGA-based Compaction Engine for Accelerating LSM-tree Key-Value Stores," *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, Dallas, TX, USA, 2020, pp. 1261-1272, doi: 10.1109/ICDE48307.2020.00113.

[9] Zhang T, Wang J, Cheng X, et al. FPGA-Accelerated Compactions for LSM-based Key-Value Store[C]. FAST. 2020: 225-237.

[10] Lim M, Jung J, Shin D. LSM-Tree Compaction Acceleration Using In-Storage Processing[C]. 2021 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia). IEEE, 2021: 1-3.

[11] LevelDB [EB/OL]. https://dbdb.io/db/leveldb/